Estrategias de detección de ransomware de cifrado

Iris Castañaga[1], Fabián Gibellini[1], Pablo Frias[1], Lorena Ruhl[1],
Leonardo Ciceri[1], Germán Parisi[1], Milagros Zea[1], Federico Bertola[1], Paula Olmedo[1]

[1] Departamento de Sistemas,
Universidad Nacional Córdoba – Facultad Regional Córdoba,
Maestro López S/N,
Córdoba Capital, Argentina
{iris.gastanaga, fabiangibellini,
pablosfrias, analialorenaruhl, leito.cba10, milyzc,
federicojbertola, paulabeatrizolmedo}@gmail.com

Resumen. Un ransomware es un malware que solicita un rescate a través de una extorsión, por ejemplo, bloquear el acceso a los recursos de la computadora, cifrar la información útil para el usuario o engañar con mensajes amenazantes. En este trabajo nos centraremos en aquellos ransomware que cifran información y piden dinero para descifrarla, ya que son los más comunes. Uno de los más conocidos es WannaCry, que tuvo un alcance mundial. Si bien la solución apropiada es realizar backup de la información cotidianamente, esto no evita que luego de un ataque se pierda tiempo y dinero en la restauración de los datos. Por lo tanto, en este trabajo, se proponen diferentes estrategias para detectar en tiempo real la ejecución de un ransomware de cifrado con el fin de destruirlo lo antes posible. Se propone una implementación en Linux usando un módulo cargable en el kernel.

1 Introducción

Un ransomware es un malware que inhibe el uso de una computadora a un usuario y pide recompensa para que pueda ser usada. Si bien son conocidos desde fines de los 80 [1], actualmente es uno de los ataques cibernéticos más frecuentes, entre los que se pueden mencionar CryptXXX [2], Petya [3] y SamSam [4]. Este último, llegó a recaudar alrededor de 6 millones de dólares entre 2016 y 2018.

Uno de los ataques más conocidos de ransomware y que afectó a más de la mitad del mundo fue el WannaCry en septiembre del 2018 [5]. Este, específicamente, es un crypto-ransomware. Estos tienen la característica que cifran la información de la víctima y luego solicitan un rescate, normalmente dinero, para que la misma sea descifrada. [6]. El blanco del WannaCry fueron todos aquellos servidores Windows que les faltó una actualización de seguridad brindada meses antes del ataque.

En el presente trabajo nos centraremos en definir estrategias para la detección de los crypto-ransomware. La información es cifrada con una clave que solo la conoce el

creador del malware, por lo que en principio la única manera de recuperar la información es pagando el rescate. Existen medidas preventivas como la realización de backup de información que permitirían resolver el problema [7]. Sin embargo, esto no evita que luego de un ataque de estas características se pierda tiempo y dinero en la restauración de los datos. Es por esto que se pretende plantear una estrategia para desarrollar un software de defensa a fin de que pueda reconocer que un proceso es un ransomware y así bloquearlo.

Este trabajo forma parte del Proyecto Homologado "Sistema de Detección de Código malicioso - ransomware", cuyo código es SIUTNCO0004991, llevado a cabo en la UTN - FRC, cuyo objetivo es "Desarrollar un sistema de detección del malware ransomware durante su ejecución en una computadora víctima y detener su avance antes de cifrar todos los archivos, basado en metodologías abiertas para identificar y analizar sus vulnerabilidades".

Las estrategias que se plantean son de carácter reactivo, es decir, la detección de un ransomware será cuando este se encuentre en ejecución, el cual, llegaría a cifrar a algunos archivos pero la intención es que la mayoría de ellos no. Para poder reconocer los patrones de un ransomware se hicieron pruebas, las cuales fueron implementadas en un ambiente aislado y controlado, tal como lo explica Elisan [8]. Además, para mejorar la inspección de cada malware se desarrolló una plataforma de código libre publicada en GitHub [9].

Se empezará por presentar y extender el concepto de función de malicia expuesto por Kharraz y Kirda [10]. Luego, se presentarán diferentes estrategias desde la más ingenua hasta la más sofisticada, las cuales se basan en el concepto extendido de la función de malicia. Lo siguiente, será profundizar en la implementación de una de las estrategias planteadas aquí, utilizando Linux como entorno de prueba. Para finalizar el trabajo se presenta una discusión sobre las diferencias entre la estrategia implementada aquí y la presentada por Kharraz y Kirda [10].

2 Función de malicia extendida

2.1 Función para calcular el puntaje de malicia

Kharraz y Kirda presentan el concepto de función de malicia y este trabajo partirá de esa definición.

Se define una función F, llamada "función de malicia", para un proceso X, a aquella función que asigna un puntaje S a un proceso cada T milisegundos de ejecución de ese proceso. Es decir:

$$F(X) = S; \text{ es una función que se recalcula cada T ms de ejecución.} \quad (1)$$

De esta manera, si el puntaje supera un umbral p significa que podemos inferir que el proceso es un ransomware, por lo que se procede a suspenderlo e informar al usuario para que tome una decisión.

$$F(X) = S \geq p; \text{ entonces se trata de un posible ransomware.} \quad (\mathbf{2})$$

2.2 Condiciones para una función de malicia

Para mejorar la definición de la función de malicia, planteada por Kharraz et al, nos llevó a plantear qué condiciones debería cumplir dicha función. Las cuales se listan a continuación.

1. El resultado de la función no debe depender del hardware.
2. Se está dispuesto a perder el 0,01% de los datos. Por ejemplo, para 1 TiB de información, o sea 1048576 MiB, la función deberá retornar un valor superior al umbral luego de que se hayan cifrado como máximo 105 MiB. Esto implica que una parte de los datos se van a cifrar pero debería ser mínima respecto a la totalidad, logrando que el proceso de restauración de backup sea mucho más rápido.
3. Es condición necesaria y suficiente que si la función retorna un resultado superior al umbral, ese proceso debe suspenderse y esperar acción del usuario.
4. Se prefiere falsos positivos antes que falsos negativos. Es decir, se prefiere detectar programas que no son ransomware antes que dejar que un ransomware no sea detectado.

Además de las condiciones anteriormente mencionadas, se plantean dos condiciones más, que sería ideal que la función de malicia las cumpla pero debido a su alta complejidad son difíciles de lograr:

- La función no debe equivocarse ante procesos que realizan copias de seguridad.
- La función no debe equivocarse ante procesos que realizan compresión de archivos.

3 Estrategias

3.1 Estrategia número 1: Utilización indiscriminada de llamadas read y write

Una llamada al sistema (o syscall) es una función que permite a un proceso que se está ejecutando en el espacio de usuario solicitar un servicio al kernel [11]. Un ransomware al leer y escribir archivos cifrados solicita acceso al disco continuamente. Esto permite inferir que este tipo de malware ejecutará muchas llamadas a read y a write. Estas llamadas solicitan al kernel poder leer y escribir información desde y hacia un archivo. Normalmente este archivo se encuentra en un disco duro.

Cada sistema operativo define sus propias llamadas al sistema, aunque existen algunas normas como POSIX [12] [13] que establecen una interfaz estándar entre el espacio de kernel y el espacio de usuario, es decir, definen una lista de llamadas al sistema. Sin embargo, tanto si el sistema operativo cumple o no con POSIX, hay 2 llamadas al sistema que de alguna u otra manera son implementadas y éstas son read y write. Por lo que las estrategias que se plantean en este trabajo asumirán que estas llamadas al sistema existen.

Se define a la función de malicia como la suma de la cantidad de llamadas a read y write. Matemáticamente:

$$F(X) = COUNT(READ, X) + COUNT(WRITE, X) \quad (3)$$

Siendo X un proceso cualquiera.

Léase a COUNT(READ, X) como la cantidad de llamadas a read realizadas por el proceso X desde que se inició.

En la Tabla 1 se puede apreciar esta idea:

Tabla 1. Cantidad indiscriminada de llamadas *read* y *write* realizadas por procesos Pi; con i = 1,2,...8.

P1	P2	P3	P4	**P5**	P6	P7	P8
10	15	20	12	**78542**	452	1200	2700

Donde en la fila superior se encuentran los procesos, y en la fila inferior, se encuentran la cantidades de llamadas realizadas por cada uno de ellos desde que se iniciaron. Como se puede observar, el proceso P5 ha realizado muchas llamadas a read y write. En principio, se puede presumir que se está tratando de un ransomware. Sin embargo, es difícil discernir si se trata de llamadas de lectura y escritura o sólo puede ser que sólo se esté tratando de lectura, es decir, de llamadas a read.

3.2 Estrategia número 2: Utilización discriminada de llamadas read y write

Para resolver el problema de la estrategia anterior, se propone que por cada proceso se creen dos contadores, uno para las llamadas a read y otro para las llamadas a write. Entonces, deberían existir dos funciones de malicia:

$$F1(X) = COUNT(READ, X) \text{ y} \quad (4)$$

$$F2(X) = COUNT(WRITE, X)$$

Siendo X un proceso cualquiera. Donde ambas funciones al mismo tiempo deben superar el umbral *p*.

La Tabla 2 muestra un ejemplo de esta estrategia:

Tabla 2. Cantidad discriminada de llamadas *read* y *write* realizadas por procesos Pi, donde i = 1,2,..8.

P1		P2		P3		P4		**P5**		P6		P7		P8	
6	4	7	8	9	11	7	5	**7800**	**9020**	402	50	300	900	2000	700

Esta estrategia parece razonable, sin embargo, después de unas cuantas pruebas se pudo comprobar que varios procesos no maliciosos que se ejecutan durante un largo tiempo también pueden llegar a números elevados de llamadas a read y a write. Sin embargo, sucede que estos procesos no maliciosos no solo ejecutan en gran cantidad llamadas a read y write sino que ejecutan muchas otras más. Esto nos lleva a pensar que realmente lo que importa es la proporción entre las llamadas a read y a write respecto a las otras. Es decir, se debe cumplir:

$$F1(X) = COUNT(READ, X) \div COUNT(ALL, X) \geq 0{,}40 \quad (5)$$

$$F2(X) = COUNT(WRITE, X) \div COUNT(ALL,X) \geq 0{,}40$$

Ambas funciones deben superar el umbral 0,4. Esto quiere decir, que tanto las llamadas a read son el 40% de todas y que las llamadas a write también lo son. Dejando solo un 20% para el resto de las llamadas. En un principio el umbral elegido fue de 0,35 pero luego de múltiples pruebas empíricas, en el entorno aislado y controlado mencionado anteriormente, se modificó a 0,4. Se puede utilizar la plataforma publicada en GitHub [9] para realizar estas pruebas empíricas. La plataforma ya incorpora algunos ejemplos de ransomware pero pueden agregarse todos los que se consideren necesarios.

Analizando los resultados de aplicar esta estrategia se observó que existen procesos como los generadores de backup o aquellos que comprimen información que serían detectados como maliciosos y realmente no lo serían. A continuación se muestran dos maneras de resolver este problema:

- A través de una lista blanca de procesos: Como son pocos los procesos que tienen el comportamiento similar al de un ransomware y uno a priori los conoce, se puede definir una lista de procesos que no serían monitoreados por el detector.
- Mejorando esta estrategia no haciendo uso del principio de preservación de longitud de los algoritmos criptográficos [14].

3.3 Estrategia número 3: Llamadas read y write con tamaño de datos leídos y escritos

Las primeras dos estrategias solo hacen uso de la cantidad de llamadas al sistema. Para tener mayor información se puede incorporar otro parámetro, por ejemplo, la cantidad de bytes utilizados en estas llamadas.

Existen algoritmos de cifrados que preservan la longitud de datos (Length-preserving encryption) [15]. Esto quiere decir, que la longitud de la entrada de datos es la misma que la de la salida. Sin embargo, muchos otros algoritmos de cifrado no preservan la longitud. Esto quiere decir, que la salida será más grande que la entrada.

La Tabla 3 de procesos puede verse como sigue:

Tabla 3. Cantidad discriminada de llamadas read y write en conjunto con el tamaño de datos leídos y escritos por procesos Pi, donde i = 1,2,...8.

P1		P2		P3		P4		**P5**		P6		P7		P8	
6	4	7	8	9	11	7	5	**39271**	**39271**	402	50	300	900	2000	700
.	.	.	.	.	.	.	.	**3.7 MB**	**3.9 MB**	.	.	.	.	.	.

Ahora se analiza la cantidad de datos leídos y la cantidad de datos escritos. Se puede sospechar que en la mayoría de los ransomware, la cantidad de datos escritos es mayor a la cantidad de datos de lectura. Y si a pesar que el ransomware utilice un algoritmo de cifrado que preserve la longitud, es muy probable que el tamaño de escri-

tura sea mayor que el de lectura debido a que se suele almacenar la clave simétrica de cifrado en el mismo archivo, por supuesto, muy probablemente la misma estará cifrada con la clave pública del atacante.

La función de malicia planteada para esta estrategia es:

$$F(X) = (W * V) / \sum_{i=0}^{n} W_i \tag{6}$$

$$F(X) = (W_1 * V_1 + W_2 * V_2 + W_3 * V_3) / \sum_{i=0}^{n} W_i$$

Donde el vector V está formado por los siguientes elementos:

- V1 = Vale 1 si la proporción de llamadas a read >= 0,40.
- V2 = Vale 1 si la proporción de llamadas a write >= 0,40.
- V3 = Vale 1 si la cantidad escrita es mayor a la cantidad de lectura

Cada elemento del vector puede tomar 1 o 0 y cada Wn son pesos definidos empíricamente.

4 Implementación

Se implementó la estrategia número 2 anteriormente presentada a través un módulo cargable en el kernel de Linux. Un módulo del kernel es un archivo que contiene código que extiende al código del kernel mientras este último está en ejecución. Es decir, no hace falta ni recompilar y ni reiniciar el sistema operativo para añadir este módulo.

Por supuesto, la ejecución de un módulo kernel se da en el ring 0 o también conocido como espacio de kernel. La idea es que este módulo mantenga por cada proceso en ejecución la tabla 4 con la siguiente información:

Tabla 4. Tabla que mantiene y contabiliza el módulo kernel por cada proceso en ejecución.

Cantidad de llamadas a write	Cantidad de llamadas a read	Cantidad de otras llamadas

Para poder actualizar la información de los contadores se necesita que el módulo intercepte cada una de las llamadas al sistema. La interceptación actualiza los contadores y ejecuta la llamada al sistema original.

Esta interceptación se realiza cuando el módulo inicia. Para ello busca a dónde se encuentran las llamadas al sistema en la memoria del kernel. Para realizar esto hace

uso de la estructura sys_call_table que tiene un puntero a cada una de las llamadas. Luego, modifica el puntero original de la llamada para que apunte a una función propia del módulo. Es decir:

```
original_write = (void *) syscall_table[__NR_write];
syscall_table[__NR_write] = &new_write;
```

Se puede notar que se almacena la llamada al sistema write en una variable global al módulo y se apunta a una nueva función new_write que está definida en el mismo módulo. En esta función es donde se actualiza los contadores. El código se ve algo similar a:

```
int new_write (unsigned int x, const char __user *y, size_t size) {

    /* Se obtiene el PID del proceso actual */
    int pid = current->pid;

    /* Se realizan las actualizaciones de los contadores del proceso actual */
    actualizacionContadores(pid);

    /* Calcular función de malicia para este proceso */
    scoreWrite = calcularFuncionMaliciaParaWrite(pid);

    /* Verificar si la función de malicia supera el umbral */
    if (scoreWrite >= umbral && scoreRead >= umbral) {
            bloquearProceso(pid);
    }

    /* Se llama a la función original */
    return original_write(x, y, size);
}
```

Tener en cuenta que se está calculando la función de malicia ante cada llamada al sistema diferente a lo definido ya que se había previsto que la misma se calcule cada

ciertos milisegundos. Sin embargo, esta pequeña variación permite realizar una implementación mucho más fácil.

5 Resultados

La estrategia número 2 funciona bastante bien como se puede apreciar en la tabla 5, la cual presenta el valor retornado por la función de malicia para varios procesos.

Tabla 5. Resultados de pruebas utilizando la estrategia número 2 y distintos procesos, sobre la plataforma para contar syscalls.

Proceso	Write		Read		Otras llamadas		Total
	Cant	F2	Cant	F1	Cant	Prop	
Ransomware 1	97862	0,495	98062	0,496	1522	0,007	197446
Ransomware 2	1565638	0,498	1565945	0,498	9409	0,002	3140992
Reproductor de música	59886	0,086	57371	0,082	576320	0,83	693577
Buscar archivo (*)	991761	0,311	10	0	2194202	0,688	3185973
Compresión (**)	4550	0,5	3912	0,429	637	0,07	9099

(*) Notar que el proceso de búsqueda de un archivo tiene muchas llamadas a write. Esto en realidad es porque se usó el comando find de Linux y que las mismas son para imprimir por pantalla cada uno de los archivos en los que va buscando. Una mejora adicional al detector sería ignorar las llamadas a write que son para escribir en pantalla y las llamadas a read que son para leer desde el teclado.

(**) En este caso, el proceso de compresión también supera los umbrales tal cual lo haría un ransomware. Con la estrategia implementada no se puede diferenciar, por lo que sería un falso positivo.

A continuación, en la Tabla 6, se mostrará que la cantidad de llamadas a write y a read no depende de la cantidad de datos a cifrar por el ransomware y las proporciones se mantienen.

Tabla 6. Resultados de pruebas utilizando la estrategia número 2 con tamaños de datos distintos, sobre la plataforma para contar syscalls

Tamaño (MiB)	Cantidad de write()	F2	Cantidad de read()	F1	Total
25	6219	0,445	6827	0,488	13976
50	12319	0,47	12927	0,493	26207
100	24519	0,483	24977	0,493	50667
200	49019	0,492	49527	0,497	99586
400	97919	0,496	98427	0,498	197447

6 Discusión

La función de malicia planteada en este trabajo cumple con las condiciones establecidas bajo cierto contexto. Las condiciones 1 y 3 son fáciles e intuitivas de verificar su cumplimiento. Las condiciones 2 y 4 son complejas de demostrar que se cumplan para todos los casos, sin embargo, nuestro trabajo las cumple en la mayoría de los casos.

Por otro lado, existen otros autores que formularon funciones de malicia para la detección de ransomware en tiempo real, por ejemplo, Kharraz y Kirda en su trabajo [10] plantean que la función de malicia debe calcularse a través de seis factores, de los cuales el más similar al tratado en este trabajo es el sexto. Ellos plantean una medida del tiempo entre escrituras pero no tienen en cuenta la lectura. Los demás factores que plantean son interesantes pero no consideramos que puedan generar una mejor detección e incluso ellos tampoco demuestran que así lo fueran. También Julian Wolf [16] presenta una serie de técnicas de detección pero no están numéricamente expresadas a través de una función de malicia.

La función de malicia planteada aquí tiene muchos interrogantes y problemas pero es una primera aproximación a la resolución del problema, y para ser una primera aproximación funciona muy bien en casos prácticos. Esta técnica de detección a través de detectar patrones en las llamadas al sistema puede extenderse para detectar otros tipos de malwares, en la sección siguiente se profundizará ello.

7 Futuros trabajos

Se pretende implementar la estrategia 3 para lograr diferenciar entre procesos que siguen un patrón similar al de un ransomware pero no son maliciosos, como procesos de copias de seguridad o procesos que realizan compresión de archivos. Incorporar la cantidad de bytes leídos y escritos en la implementación del cálculo de la función de malicia no es una tarea trivial y los detalles de la misma quedarán para otro trabajo.

De este trabajo surge la idea de desarrollar un framework para la detección de malwares a través de las llamadas al sistema que realizan. Básicamente las llamadas al sistema que realiza un proceso es como una marca de lo que ejecutan. Si esas ejecuciones coinciden con patrones maliciosos se podría tomar una medida reactiva. Un malware de tipo fileless [17], es aquel que no existe en el disco duro y suele descargar su código fuente desde internet y ejecutarlo en memoria. Detectar esto realmente es muy complejo para cualquier herramienta de defensa. Con la técnica de detección de patrones de llamadas al sistema uno podría generar patrones maliciosos para este tipo de procesos.

Referencias

1. Richardson R., North M. (2017). Ransomware: Evolution, Mitigation and Prevention - Ronny Richardson, Max North - Kennesaw State University. International Management Review. Vol. 13 No. 1 2017. Recuperado el 3 de mayo del 2019 de http://scholarspress.us/journals/IMR/pdf/IMR-1-2017.%20pdf/IMR-v13n1art2.pdf.
2. Lupu E., Sgandurra D., Muñoz-González M., Mohsen R. (2016). Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection. arXiv.org E-print Archive arXiv:1609.03020 [cs.CR].
3. MalwareLabs. (2017). Petya – Taking Ransomware To The Low Leve).
4. Sophos. (2018). SamSam: The (Almost) Six Million Dollar Ransomware. Recuperado el 3 de mayo del 2019 de https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/SamSam-The-Almost-Six-Million-Dollar-Ransomware.pdf.
5. National Audit Office de Reino Unido. (2018). Investigation: WannaCry cyber attack and the NHS. Recuperado el 3 de mayo del 2019 de https://www.nao.org.uk/wp-content/uploads/2017/10/Investigation-WannaCry-cyber-attack-and-the-NHS.pdf.
6. Sanggeun S., Bongjoon K., Sangjun L. (2016). The effective Ransomware Prevention Technique Using Process Monitoring on Android Platform. Soongsil University.
7. Young A., Yung M. (1996). Cryptovirology: extortion-based security threats and countermeasures. In Proceedings 1996 IEEE Symposium on Security and Privacy. 129–140. Recuperado el 3 de mayo del 2019 de http://dx.doi.org/10.1109/SECPRI.1996.502676.
8. Elisan C. (2015). Part II - Malware Research Lab. McGraw-Hill. Advanced Malware Analysis. ISBN 9780071819756.
9. Plataforma para contar syscalls y analizar algunos parámetros de interés para detectar un ransomware. Enlace al código fuente https://github.com/LabSis/contador_syscalls.
10. Kharraz A., Kirda E. (2017) Redemption: Real-Time Protection Against Ransomware at End-Hosts. In: Dacier M., Bailey M., Polychronakis M., Antonakakis M. (eds) Research in Attacks, Intrusions, and Defenses. RAID 2017. Lecture Notes in Computer Science, vol 10453. Springer, Cham.

11. https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl.
12. Austin Common Standards Revision Group. (Última actualización, 2017) POSIX® 1003.1 Frequently Asked Questions (FAQ Version 1.16). Página oficial: http://www.opengroup.org/austin/papers/posix_faq.html.
13. Harbour M. (2001). POSIX de Tiempo Real. Departamento de Electrónica. Universidad de Cantabria. Santander. Recuperado el 3 de mayo del 2019 de https://www.ctr.unican.es/publications/mgh-1993b.pdf.
14. Goldreich O. (2001). Foundations of Cryptography – Basic Tools, Cambridge University Press. P.39.
15. Jianwu Z., Hui L., Mingsheng L. (2007) On Length-Preserving Symmetric Cryptography. In: Sobh T., Elleithy K., Mahmood A., Karim M. (eds) Innovative Algorithms and Techniques in Automation, Industrial Electronics and Telecommunications. Springer, Dordrecht.
16. Wolf J. (2017). Ransomware Detection. Friedrich-Alexander-University.
17. (Octubre, 2018). El boom de los ataques de malware sin fichero: ¿cómo combatirlo? Panda Security Info. Recuperado el 3 de mayo del 2019 de https://www.pandasecurity.com/spain/mediacenter/seguridad/boom-ataques-fileless-malware/